

*PyDX 2016*

---

# Advanced Git

David Baumgold  
@singingwolfboy

---

*[bit.ly/git-pydx-2016](http://bit.ly/git-pydx-2016)*

---

# The Story So Far

---

\$ `git clone <url>` Download the repo

\$ `git checkout <branch>` Switch to a feature branch

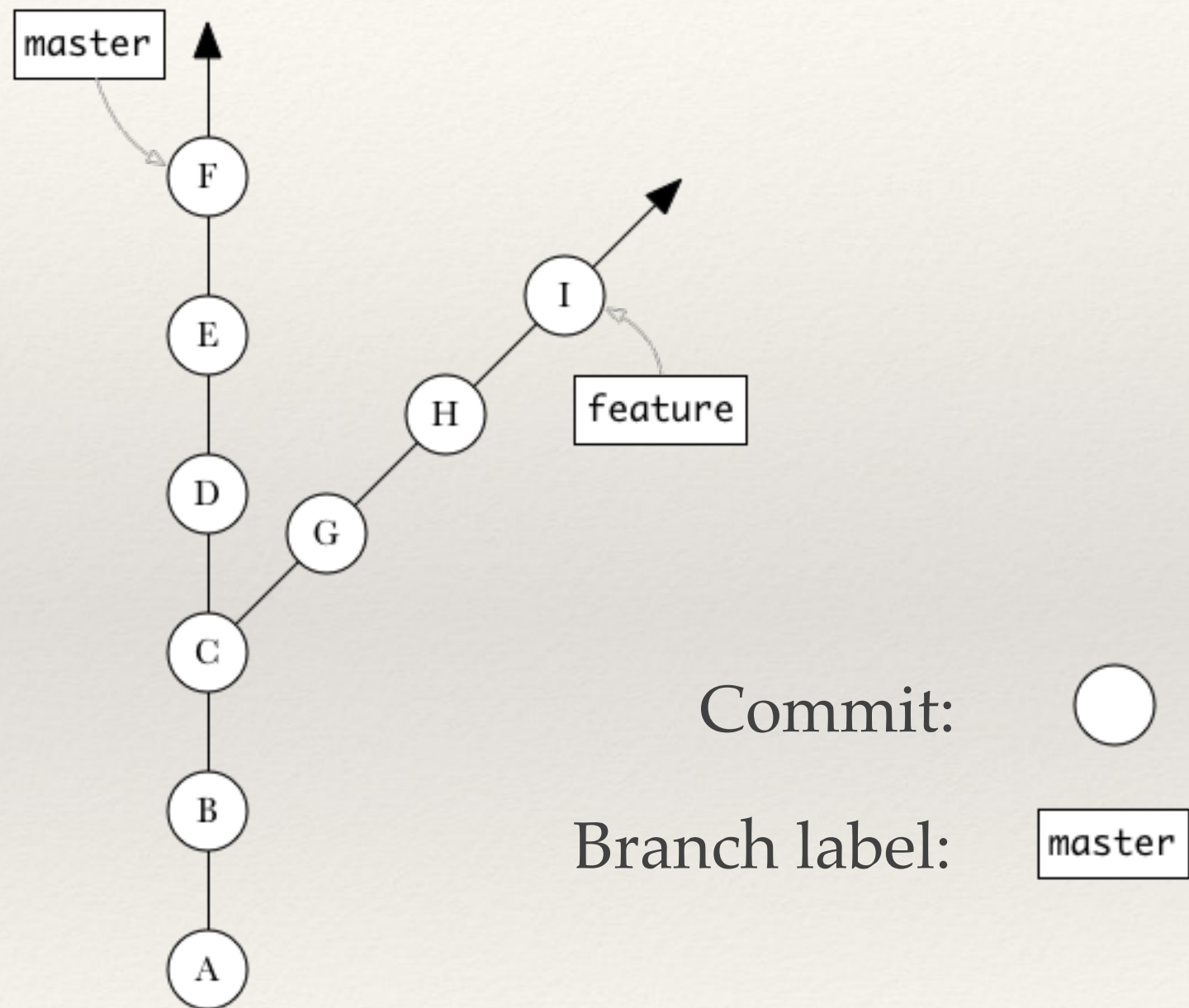
Edit some files

\$ `git commit` Commit your changes

\$ `git push/pull` Sync changes to GitHub

Merge pull request on GitHub

# Visual Terminology





---

# Table of Contents

---

Preface

`status, show`

Ch 1

`blame`

Ch 2

`cherry-pick`

Ch 3

`reset`

Ch 4

`rebase`

Ch 5

`reflog`

Ch 6

`squashing & splitting`

Ch 7

`bisect`

---

# Preface: status

---

```
$ git status  
On branch master  
nothing to commit, working directory clean
```

---

# Preface: status

---

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    deleted:    removed.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   modified.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    added.txt
```

---

# Preface: show

---

```
$ git show
commit 15f81303f58fc7d8fc8f598a8c9be94e783cced2
Author: David Baumgold <david@davidbaumgold.com>
Date:   Sun Mar 15 21:48:25 2015 -0400
```

Detailed commit message

```
diff --git a/modified.txt b/modified.txt
index 2579662..d704eff 100644
--- a/modified.txt
+++ b/modified.txt
@@ -1, +1 @@
  This line stayed the same
-This line was removed
+This line was added
```



---

# Table of Contents

---

Preface

`status, show`

Ch 1

`blame`

Ch 2

`cherry-pick`

Ch 3

`reset`

Ch 4

`rebase`

Ch 5

`reflog`

Ch 6

`squashing & splitting`

Ch 7

`bisect`



---

# Ch 1: b1ame

---

“What the...  
who wrote this code?”

```
$ git blame path/to/file.py
```

will tell you who to blame!

---

# Ch 1: b`l`ame

---

For each line of the file, b`l`ame will find the last commit to edit the line, and it will tell you:

- Commit hash
- Author's name
- Date of commit

ProTip: use `show` to look up the commit message!

```
$ git show d47312e1
```

# Ch 1: b lame

```
25ad0c5f setup.py (Sarina Canelake 2013-07-09 14:42:28 -0400 1) """Set up for XBlock"""
24c2a33d setup.py (Calen Pennington 2013-01-04 12:42:17 -0500 2) from setuptools import setup
34adc933 xblock/setup.py (Ned Batchelder 2012-11-30 15:20:57 -0500 3)
deb68879 setup.py (Calen Pennington 2014-11-10 13:33:31 -0500 4) import versioneer
deb68879 setup.py (Calen Pennington 2014-11-10 13:33:31 -0500 5) versioneer.VCS = 'git'
deb68879 setup.py (Calen Pennington 2014-11-10 13:33:31 -0500 6) versioneer.versionfile_source = 'xblock/_version.py'
deb68879 setup.py (Calen Pennington 2014-11-10 13:33:31 -0500 7) versioneer.versionfile_build = 'xblock/_version.py'
ba6d5c45 setup.py (Calen Pennington 2014-11-10 13:50:38 -0500 8) versioneer.tag_prefix = 'xblock-' # tags are like 1.2.0
ba6d5c45 setup.py (Calen Pennington 2014-11-10 13:50:38 -0500 9) versioneer.parentdir_prefix = 'XBlock-' # dirname like
'myproject-1.2.0'
deb68879 setup.py (Calen Pennington 2014-11-10 13:33:31 -0500 10)
34adc933 xblock/setup.py (Ned Batchelder 2012-11-30 15:20:57 -0500 11) setup(
34adc933 xblock/setup.py (Ned Batchelder 2012-11-30 15:20:57 -0500 12)     name='XBlock',
deb68879 setup.py (Calen Pennington 2014-11-10 13:33:31 -0500 13)     version=versioneer.get_version(),
deb68879 setup.py (Calen Pennington 2014-11-10 13:33:31 -0500 14)     cmdclass=versioneer.get_cmdclass(),
34adc933 xblock/setup.py (Ned Batchelder 2012-11-30 15:20:57 -0500 15)     description='XBlock Core Library',
d47312e1 setup.py (Ned Batchelder 2014-02-02 07:33:04 -0500 16)     packages=[
d47312e1 setup.py (Ned Batchelder 2014-02-02 07:33:04 -0500 17)         'xblock',
d47312e1 setup.py (Ned Batchelder 2014-02-02 07:33:04 -0500 18)         'xblock.django',
776c85ce setup.py (Piotr Mitros 2014-07-26 18:30:13 -0400 19)         'xblock.reference',
d47312e1 setup.py (Ned Batchelder 2014-02-02 07:33:04 -0500 20)     ],
ffe1375c setup.py (Ned Batchelder 2013-01-22 12:10:21 -0500 21)     install_requires=[
d47312e1 setup.py (Ned Batchelder 2014-02-02 07:33:04 -0500 22)         'lxml',
843f42eb setup.py (Calen Pennington 2014-12-11 08:25:51 -0500 23)         'markupsafe',
2ac249d5 setup.py (Will Daly 2014-03-13 18:20:48 -0400 24)         'python-dateutil',
843f42eb setup.py (Calen Pennington 2014-12-11 08:25:51 -0500 25)         'pytz',
843f42eb setup.py (Calen Pennington 2014-12-11 08:25:51 -0500 26)         'webob',
118d4817 setup.py (David Baumgold 2015-03-11 10:05:24 -0400 27)     ],
118d4817 setup.py (David Baumgold 2015-03-11 10:05:24 -0400 28)     license='Apache 2.0',
118d4817 setup.py (David Baumgold 2015-03-11 10:05:24 -0400 29)     classifiers=(
118d4817 setup.py (David Baumgold 2015-03-11 10:05:24 -0400 30)         "License :: OSI Approved :: Apache Software License 2.0",
118d4817 setup.py (David Baumgold 2015-03-11 10:05:24 -0400 31)     )
34adc933 xblock/setup.py (Ned Batchelder 2012-11-30 15:20:57 -0500 32) )
```



---

# Ch 1: blame

---

```
25ad0c5f  setup.py          (Sarina Canelake
24c2a33d  setup.py          (Calen Pennington
34adc933  xblock/setup.py    (Ned Batchelder
deb68879  setup.py          (Calen Pennington
deb68879  setup.py          (Calen Pennington
```

```
2013-07-09 14:42:28 -0400 1) """Set up for XBlock"""
2013-01-04 12:42:17 -0500 2) from setuptools import setup
2012-11-30 15:20:57 -0500 3)
2014-11-10 13:33:31 -0500 4) import versioneer
2014-11-10 13:33:31 -0500 5) versioneer.VCS = 'git'
```

---

# Table of Contents

---

Preface

`status, show`

Ch 1

`blame`

Ch 2

`cherry-pick`

Ch 3

`reset`

Ch 4

`rebase`

Ch 5

`reflog`

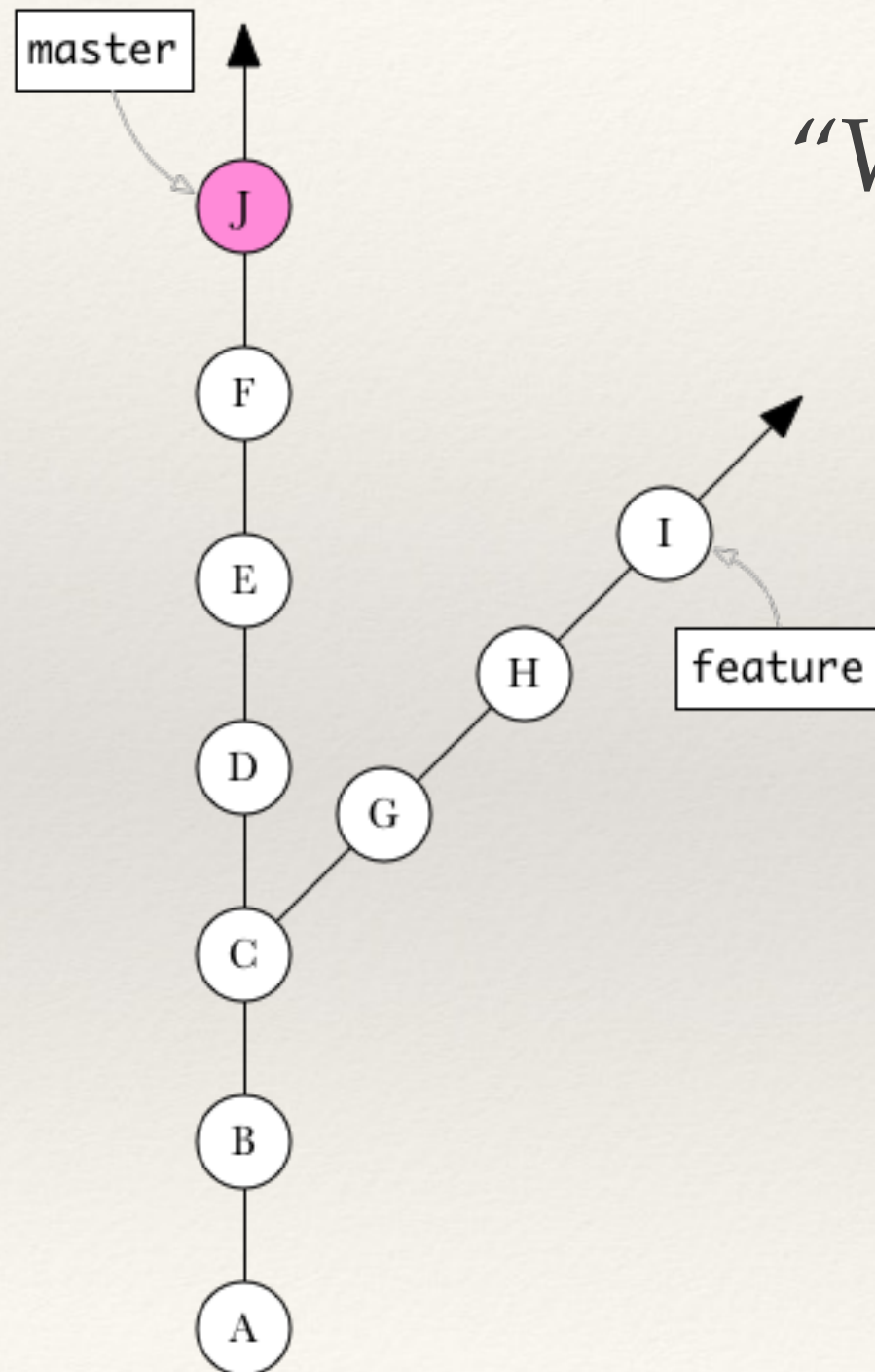
Ch 6

`squashing & splitting`

Ch 7

`bisect`

# Ch 2: cherry-pick



“Whoops, I committed to master when I meant to commit to my feature branch. I need to move my commit!”



---

# Ch 2: cherry-pick

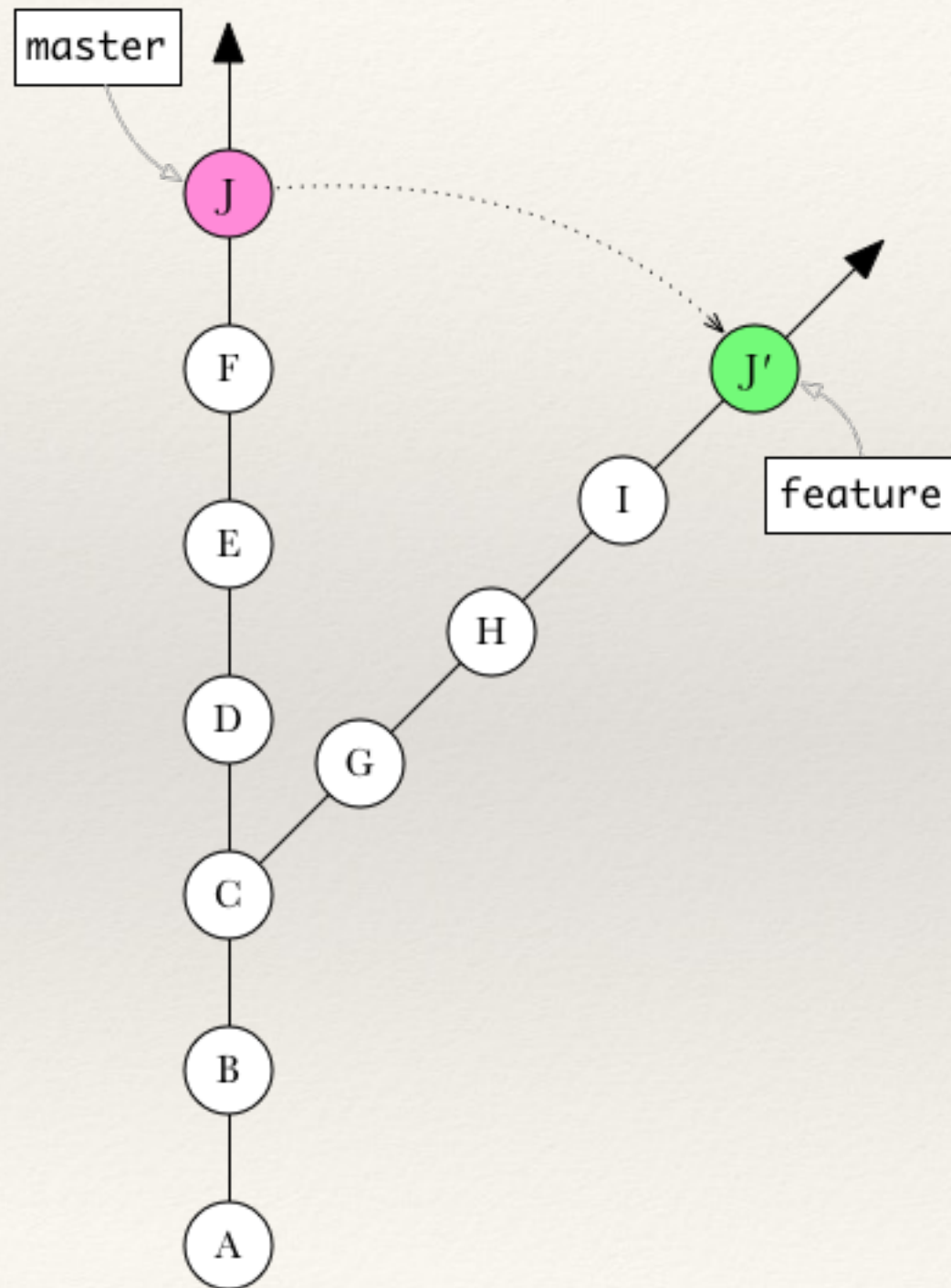
---

```
$ git show
commit 1d5b2e2b273dbb945c6bf5e541d5f1c725ac906d
# ... ignore the rest ...

$ git checkout feature
Switched to branch 'feature'

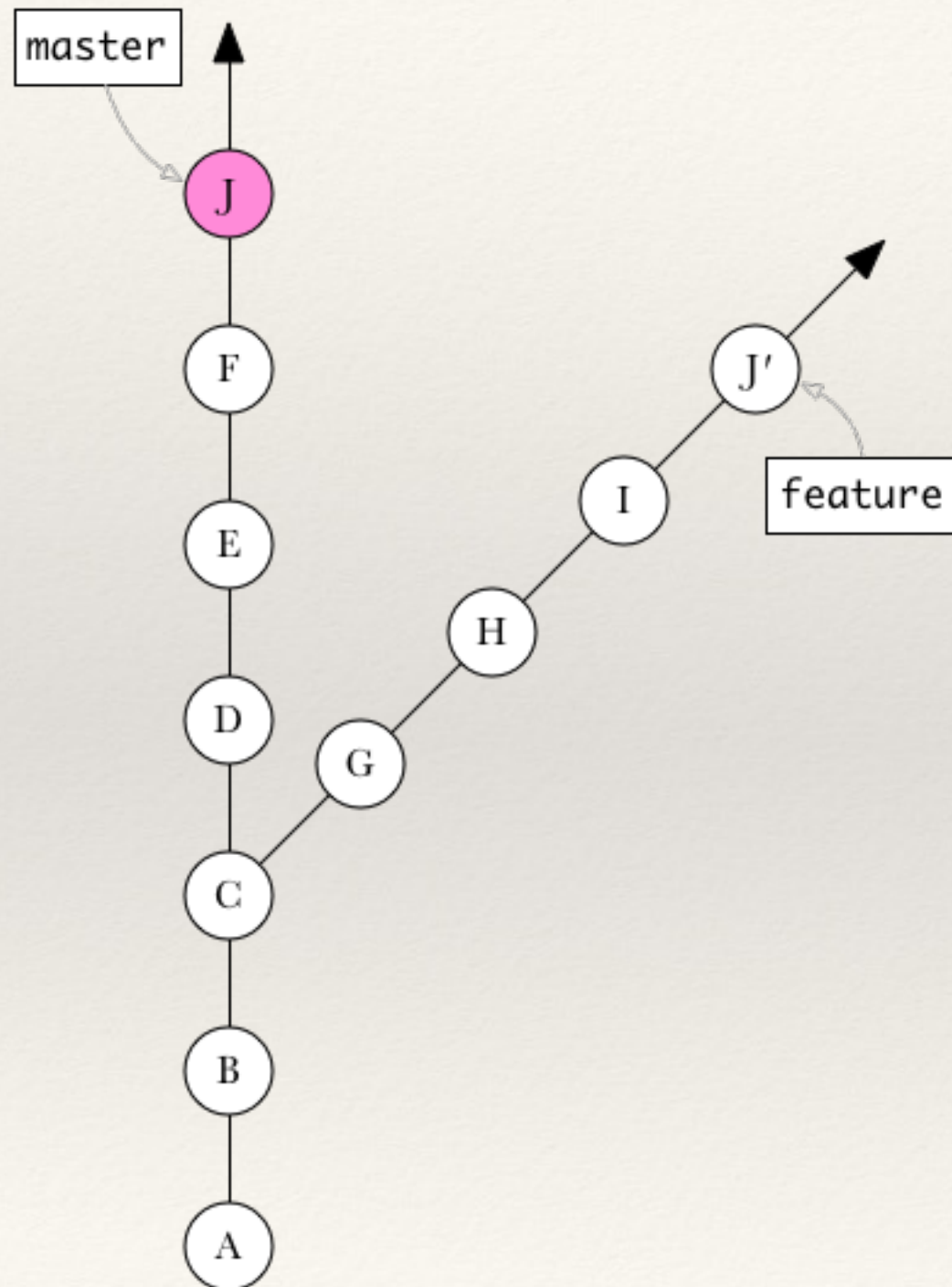
$ git cherry-pick 1d5b2e
[master 8b8d32c] original commit message
Date: Sun Mar 15 22:04:48 2015 -0400
1 file changed, 1 insertion(+)
```

# Ch 2: cherry-pick



cherry-pick  
creates an  
**entirely new commit**  
based off the original,  
and it  
**does *not* delete**  
**the original commit**

# Ch 3: reset



“Alright, how do I  
remove J from master?”



---

# Ch 3: reset

---

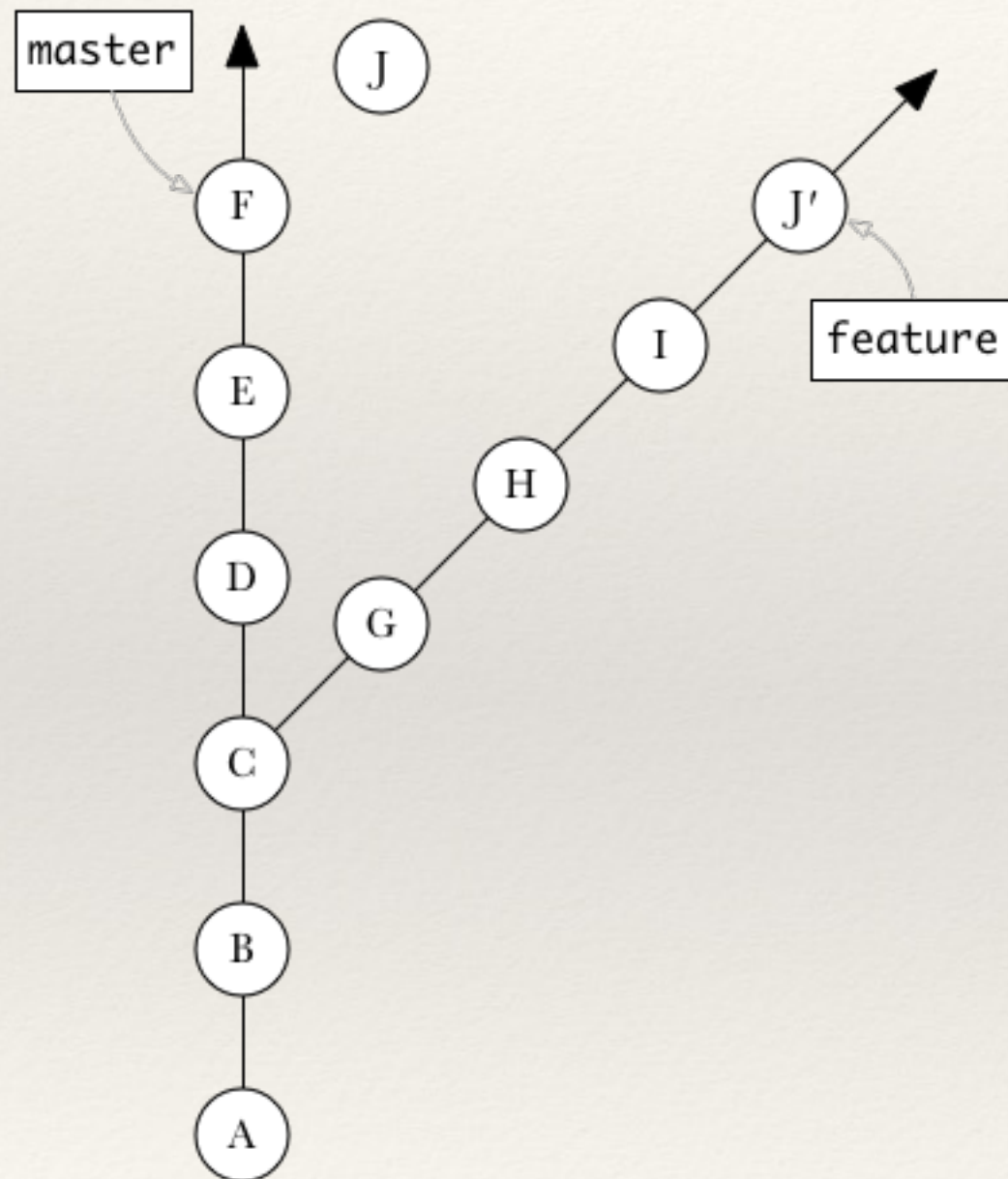
```
$ git checkout master  
Switched to branch 'master'  
  
$ git reset --hard HEAD~  
HEAD is now at 15f8130 made things work
```

HEAD == “the commit I’m currently sitting on”

HEAD~ == “this commit’s parent”

HEAD~~ == “this commit’s grandparent” (etc)

# Ch 3: reset



reset reassigns  
the branch pointer

J will get cleaned up  
by git's garbage collector  
eventually

---

# Table of Contents

---

Preface

status, show

Ch 1

blame

Ch 2

cherry-pick

Ch 3

reset

Ch 4

rebase

Ch 5

reflog

Ch 6

squashing & splitting

Ch 7

bisect



---

# WARNING

---

rebase is a command for changing history!  
Use its awesome power responsibly!

wibbly  
wobbly  
timey  
wimey



is no fun  
when you're on  
the receiving end.

---

# WARNING

---

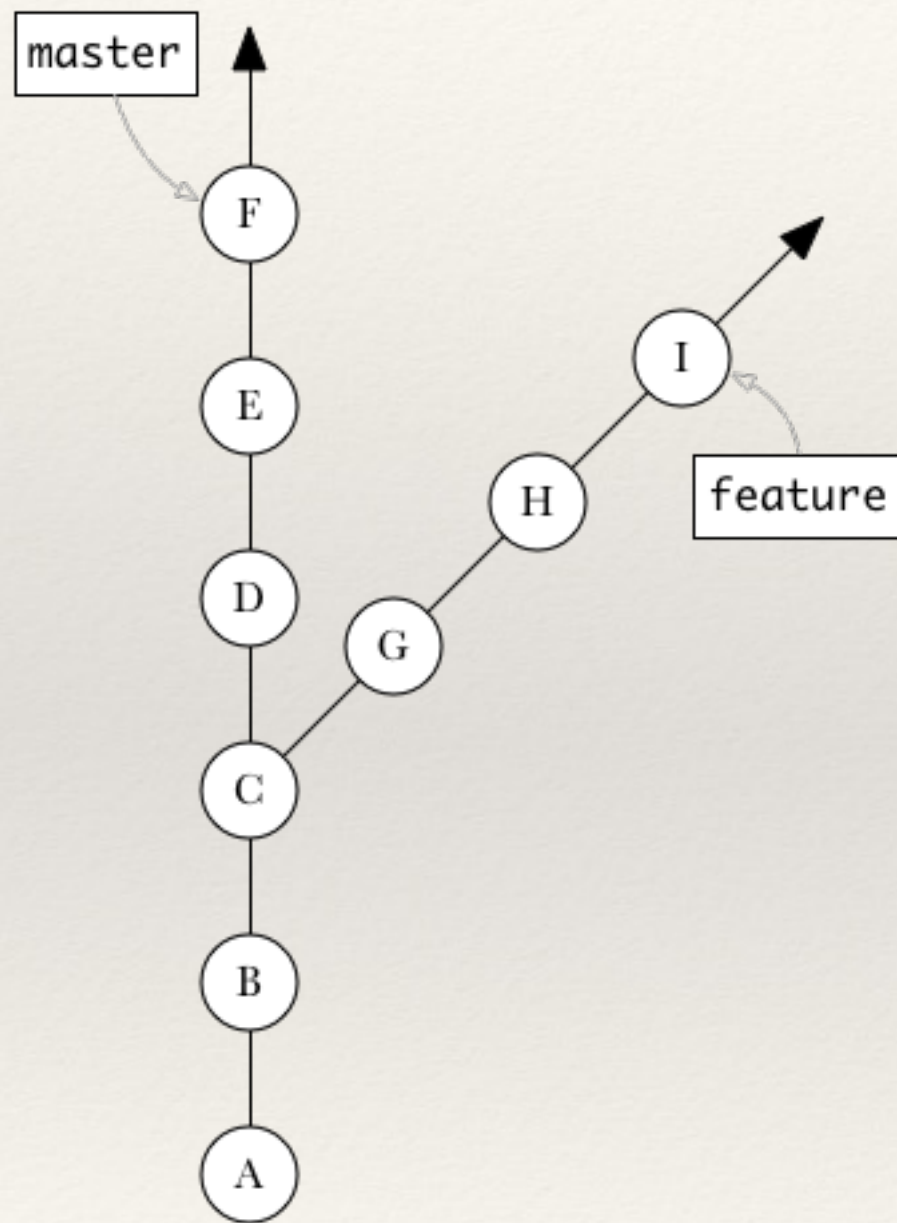
Never change history when other people might be using your branch, unless they know you're doing so.

Never change history on master.

Best practice: only change history for commits that have not yet been pushed.



# Ch 4: rebase



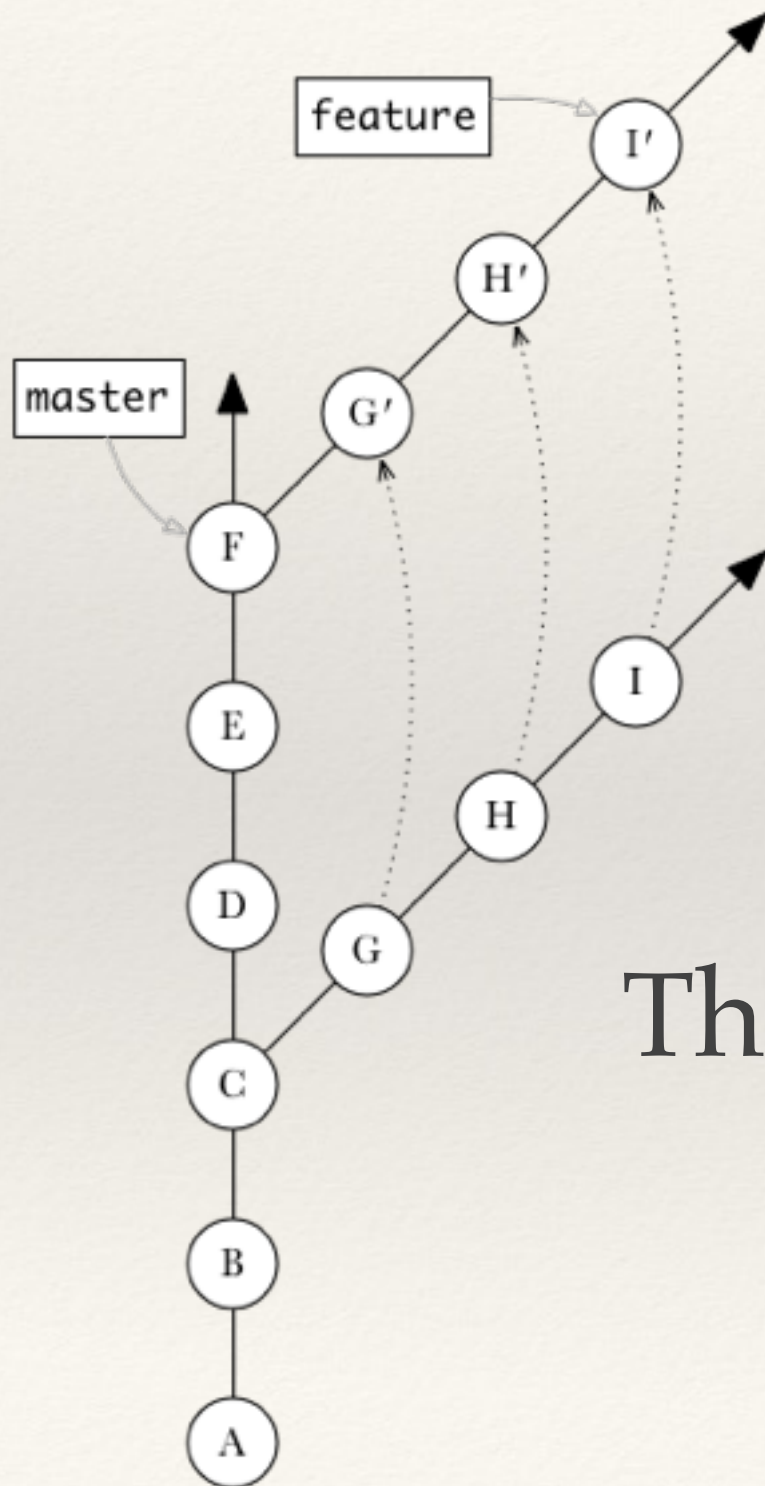
“master has changed since I started my feature branch, and I want to bring my branch up to date with master.

What’s the best way to do that?”

Don’t merge — rebase!



# Ch 4: rebase



- Finds the merge base
- Cherry-picks all commits
- Reassigns the branch pointer

The branch has a new base —  
it has been re-based!

---

# Ch 4: rebase

---

```
$ git checkout feature
Switched to branch 'feature'

$ git rebase master
First, rewinding head to replay your work on top of
it...
Applying: Added B.txt
Applying: Added another line for B.txt
Applying: Added a third line for B.txt
```

---

# Ch 4: rebase

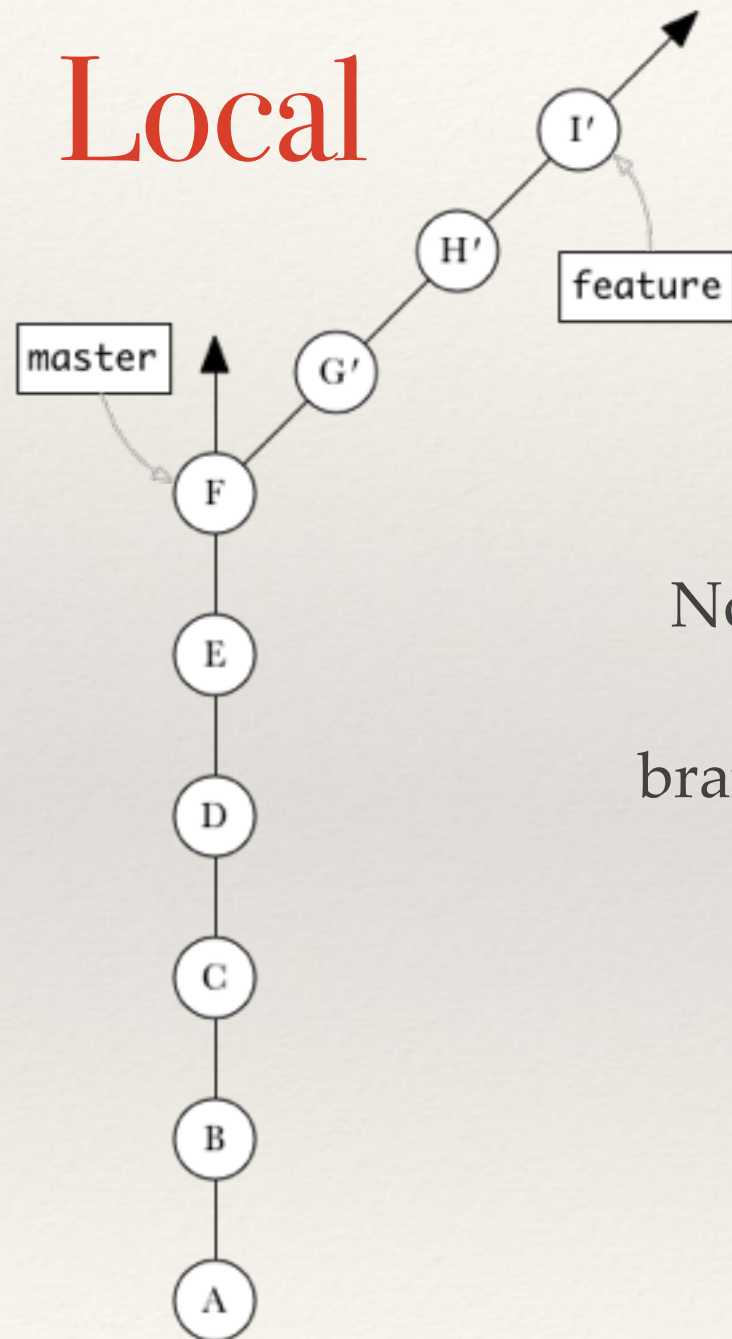
---

```
$ git status
On branch feature
Your branch and 'origin/feature' have diverged,
and have 6 and 3 different commits each,
respectively.
  (use "git pull" to merge the remote branch
   into yours)
nothing to commit, working directory clean
```

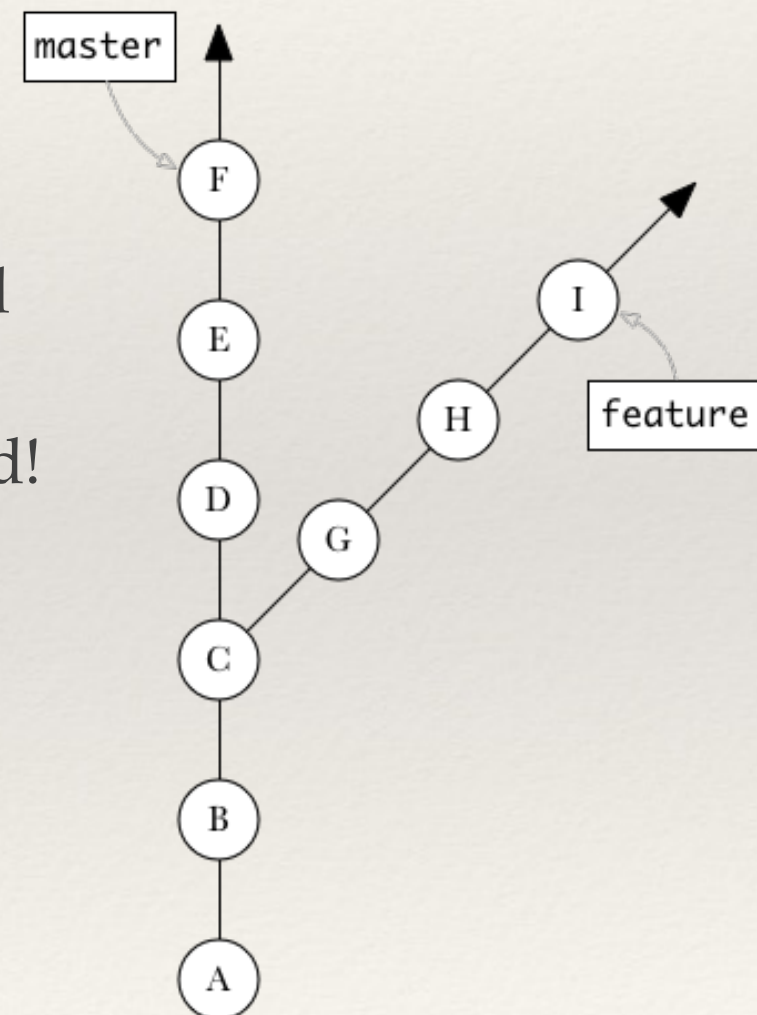


# Ch 4: rebase

Local



Remote



No way to go forward  
from I to I':  
branches have diverged!

---

# Ch 4: rebase

---

```
$ git push
To git@github.com:singingwolfboy/example.git
 ! [rejected]        feature -> feature (non-fast-forward)
error: failed to push some refs to 'git@github.com:singingwolfboy/example.git'
hint: Updates were rejected because the tip of your current branch is behind
hint: its remote counterpart. Integrate the remote changes (e.g.
hint: 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

git push is saying:

“You want me to do *what*?

But that would mean changing history!

Are you sure that’s what you want?”


---

# Ch 4: rebase

---

Use `git push -f` to force it:

```
$ git push -f
Counting objects: 9, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (7/7), done.
Writing objects: 100% (9/9), 946 bytes | 0 bytes/s, done.
Total 9 (delta 0), reused 0 (delta 0)
To git@github.com:singingwolfboy/example.git
+ dc206fd...ef6a658 feature -> feature (forced update)
```





# Ch 4: rebase

Sometimes you get conflicts...

```
$ git rebase master
First, rewinding head to replay your work on top of it...
Applying: Adding a different line to A.txt
Using index info to reconstruct a base tree...
M A.txt
Falling back to patching base and 3-way merge...
Auto-merging A.txt
CONFLICT (content): Merge conflict in A.txt
Failed to merge in the changes
Patch failed at 0001 Adding a different line to A.txt
The copy of the patch that failed is found in:
    /Users/singingwolfboy/example/.git/rebase-apply/patch
```

When you have resolved this problem, run "git rebase --continue".  
If you prefer to skip this patch, run "git rebase --skip" instead.  
To check out the original branch and stop rebasing, run "git rebase --abort".

---

# Ch 4: rebase

---

git status will show you which files are in conflict

```
$ git status
rebase in progress; onto e98d69f
You are currently rebasing branch 'conflicted' on 'e98d69f'.
  (fix conflicts and then run "git rebase --continue")
  (use "git rebase --skip" to skip this patch)
  (use "git rebase --abort" to check out the original branch)

Unmerged paths:
  (use "git reset HEAD <file>..." to unstage)
  (use "git add <file>..." to mark resolution)

    both modified:   A.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

---

# Ch 4: rebase

---

Looks the same as a merge conflict!

```
$ cat A.txt  
line one  
<<<<<< HEAD  
line two  
line three  
=====  
this line is different  
>>>>>> Adding a different line to A.txt
```



---

# Ch 4: rebase

---

But the resolution is different...

```
$ git status
rebase in progress; onto e98d69f
You are currently rebasing branch 'conflicted'
on 'e98d69f'.
  (fix conflicts and then run "git rebase --continue")
  (use "git rebase --skip" to skip this patch)
  (use "git rebase --abort" to check out the original
branch)
```

`git rebase --continue`

---

# Ch 4: rebase

---

If something's wrong, and you want to start over...

```
$ git status
rebase in progress; onto e98d69f
You are currently rebasing branch 'conflicted'
on 'e98d69f'.
(fix conflicts and then run "git rebase --continue")
(use "git rebase --skip" to skip this patch)
(use "git rebase --abort" to check out the original
branch)
```

`git rebase --abort`

---

# Ch 2: cherry-pick

---

You can get conflicts with cherry-pick, as well

```
$ git cherry-pick e98d69f0a5942704076182139acb50856ca8bc7c
error: could not apply e98d69f... Added a third line to A.txt
hint: after resolving the conflicts, mark the corrected paths
hint: with 'git add <paths>' or 'git rm <paths>'
hint: and commit the result with 'git commit'
```

`git status` is still your friend!



---

# Ch 2: cherry-pick

---

Resolve the conflicts and then

```
$ git status
On branch conflicted
You are currently cherry-picking commit e98d69f.
  (fix conflicts and run "git cherry-pick --continue")
  (use "git cherry-pick --abort" to cancel the cherry-
pick operation)
```

`git cherry-pick --continue`

---

# Ch 2: cherry-pick

---

Not worth the trouble?

```
$ git status
On branch conflicted
You are currently cherry-picking commit e98d69f.
  (fix conflicts and run "git cherry-pick --continue")
  (use "git cherry-pick --abort" to cancel the cherry-
pick operation)
```

```
git cherry-pick --abort
```

---

# Table of Contents

---

Preface

status, show

Ch 1

blame

Ch 2

cherry-pick

**throwback!**

Ch 3

reset

Ch 4

rebase

Ch 5

reflog

Ch 6

squashing & splitting

Ch 7

bisect



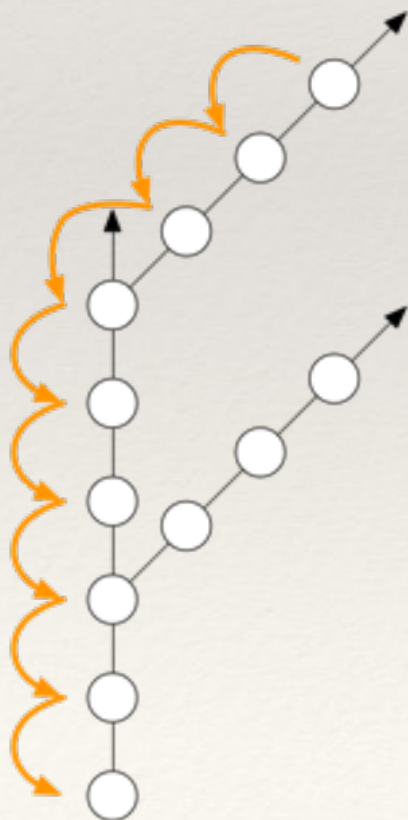
---

# Ch 5: reflog

---

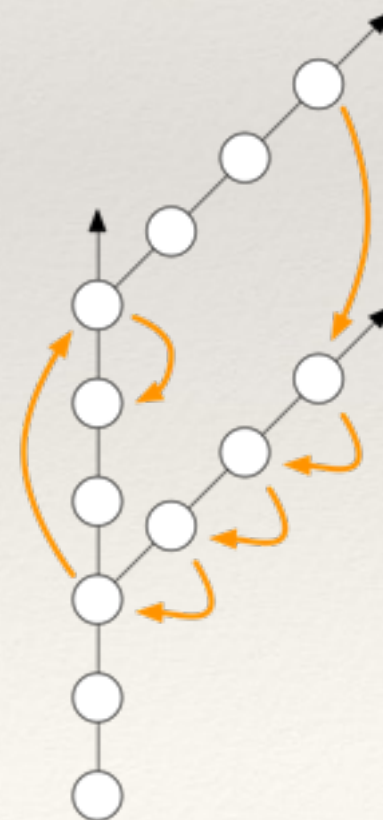
`git log`

shows commits in  
**ancestor order**



`git reflog`

shows commits in  
**order of when you  
lasted referenced them**



---

# Ch 5: reflog

---

“Oh no, I screwed up  
and I want to get back  
to the way things were before,  
but I didn’t write down  
the commit hash!”

reflog to the rescue!

---

# Ch 5: reflog

---

```
$ git reflog
909bf0d HEAD@{0}: rebase: aborting
e98d69f HEAD@{1}: rebase: checkout master
909bf0d HEAD@{2}: commit: Adding a different line to A.txt
db06ae9 HEAD@{3}: checkout: moving from db06ae99d4b6 to conflicted
db06ae9 HEAD@{4}: checkout: moving from master to db06ae99d4b6
e98d69f HEAD@{5}: checkout: moving from feature to master
ef6a658 HEAD@{6}: rebase finished: returning to refs/heads/feature
ef6a658 HEAD@{7}: rebase: Added a third line for B.txt
f581b81 HEAD@{8}: rebase: Added another line for B.txt
75f0730 HEAD@{9}: rebase: Added B.txt
e98d69f HEAD@{10}: rebase: checkout master
```

Step 1: find the commit you want



---

# Ch 5: reflog

---

Step 2: checkout the commit,  
and make sure it's what you want

```
$ git checkout 3ca7892
```

Step 3: reset the branch pointer  
back to the commit

```
$ git checkout feature  
$ git reset --hard 3ca7892
```

---

# Table of Contents

---

Preface

status, show

Ch 1

blame

Ch 2

cherry-pick

Ch 3

reset

Ch 4

rebase

Ch 5

reflog

Ch 6

squashing & splitting

Ch 7

bisect

---

# Ch 6: squashing commits

---

“Darn, I forgot to include this file  
in the commit I just made!”

```
$ git add missing-file.txt  
$ git commit --amend
```

Makes a new commit with your file added,  
and replaces the most recent commit with the new one!  
No more “added missing file” commit messages!



---

# Ch 6: squashing commits

---

“But I already have lots of commits like that!  
It’s not just my most recent commit...”

```
$ git rebase --interactive
```

Time to bring out the big guns.

---

# Ch 6: squashing commits

---

Interactive rebase needs somewhere to start.  
To look at the last 5 commits, you can use `HEAD~5`  
(or use whatever number you want)

```
$ git rebase --interactive HEAD~5
```

Git will open a file in your text editor,  
so that you can provide further instructions

# Ch 6: squashing commits

actions

commits

```
pick 11e8557 First commit!
pick e98d69f Added a widget
pick 75f0730 oops, missed a file
pick f581b81 fixed a typo
pick ef6a658 Added a second widget

# Rebase db06ae9..ef6a658 onto db06ae9
#
# Commands:
# p, pick = use commit
# r, reword = use commit, but edit the commit message
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
# f, fixup = like "squash", but discard this commit's log message
# x, exec = run command (the rest of the line) using shell
```

instructions



# Ch 6: squashing commits

actions

commits

```
pick 11e8557 First commit!
pick e98d69f Added a widget
squash 75f0730 oops, missed a file
squash f581b81 fixed a typo
pick ef6a658 Added a second widget
```

```
# Rebase db06ae9..ef6a658 onto db06ae9
```

```
#
```

```
# Commands:
```

```
# p, pick = use commit
```

```
# r, reword = use commit, but edit the commit message
```

```
# e, edit = use commit, but stop for amending
```

```
# s, squash = use commit, but meld into previous commit
```

```
# f, fixup = like "squash", but discard this commit's log message
```

```
# x, exec = run command (the rest of the line) using shell
```

instructions

---

# Ch 6: squashing commits

---

Saving and quitting your editor  
will cause it to immediately reopen

```
# This is a combination of 3 commits.
```

```
# The first commit's message is:
```

```
Added a widget
```

```
# This is the 2nd commit message:
```

```
oops, missed a file
```

```
# This is the 3rd commit message:
```

```
fixed a typo
```

so that you can write a new message  
for your single, squashed commit

---

# Ch 6: squashing commits

---

Save and quit again,  
and Git will apply  
the changes you requested.  
No more “fixed typo” commits!

**WARNING:** squashing commits changes history!  
Only do this for unpushed commits!



---

# Ch 6: splitting commits

---

“My commit is too big,  
can I split it into smaller ones?”

```
$ git rebase --interactive
```

Let's change some history.

---

# Ch 6: splitting commits

---

```
pick 21e8569 First commit!
pick 198dc9f Did a bunch of things
pick 79f0c3a Made the corners rounded

# Rebase db067e9..79f0c3a onto db067e9
#
# Commands:
# p, pick = use commit
# r, reword = use commit, but edit the commit message
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
# f, fixup = like "squash", but discard this commit's log message
# x, exec = run command (the rest of the line) using shell
```

That second commit looks too big...

---

# Ch 6: splitting commits

---

```
pick 21e8569 First commit!
edit 198dc9f Did a bunch of things
pick 79f0c3a Made the corners rounded

# Rebase db067e9..79f0c3a onto db067e9
#
# Commands:
# p, pick = use commit
# r, reword = use commit, but edit the commit message
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
# f, fixup = like "squash", but discard this commit's log message
# x, exec = run command (the rest of the line) using shell
```

So we'll use the `edit` command!



---

# Ch 6: splitting commits

---

```
$ git rebase -i HEAD~3  
Stopped at 198dc9f... Did a bunch of things  
You can amend the commit now, with
```

```
git commit --amend
```

Once you are satisfied with your changes, run

```
git rebase --continue
```

Git will pause in the rebase process,  
and give us as much time as we want  
to create some new commits

---

# Ch 6: splitting commits

---

The too-big commit is already present, so lets pop it off, but keep the changes:

```
$ git reset HEAD~
```

Note that I am *not* using `--hard`, because I want to keep the changes

---

# Ch 6: splitting commits

---

```
$ git status
rebase in progress; onto 6b44332
You are currently editing a commit while rebasing branch
'feature' on '6b44332'.
  (use "git commit --amend" to amend the current commit)
  (use "git rebase --continue" once you are satisfied with your
  changes)

Untracked files:
  (use "git add <file>..." to include in what will be committed)

file1.py
file2.py
file3.py

nothing added to commit but untracked files present (use "git
add" to track)
```



---

# Ch 6: splitting commits

---

```
$ git add file1.py
$ git commit -m "Detailed message for file1 changes"
[detached HEAD f59aaee] Detailed message for file1 changes
1 file changed, 1 insertion(+)

$ git add file2.py
$ git commit -m "Detailed message for file2 changes"
[detached HEAD ceb16d3] Detailed message for file2 changes
1 file changed, 1 insertion(+)

$ git add file3.py
$ git commit -m "Detailed message for file3 changes"
[detached HEAD 6d2930a] Detailed message for file3 changes
1 file changed, 1 insertion(+)
```

---

# Ch 6: splitting commits

---

And of course, when we're done:

```
$ git rebase --continue
```

Finish the rebase, and  
admire your cleaner commit history!

---

# Table of Contents

---

Preface

status, show

Ch 1

blame

Ch 2

cherry-pick

Ch 3

reset

Ch 4

rebase

Ch 5

reflog

Ch 6

squashing & splitting

Ch 7

**bisect**



---

# Ch 7: `bisect`

---

“The feature’s broken?  
But it was working just fine  
two months ago... what changed?”

`bisect` will help you quickly find  
the commit that introduced the problem

---

# Ch 7: `bisect`

---

You need three things to use `bisect`:

- A test to determine if things are broken (manual is OK, automated is better)
- A commit where things were working
- A commit where things are broken

`bisect` will use binary search  
to find the commit where  
things went from good to bad

---

# Ch 7: bisect

---

```
$ git bisect start  
$ git checkout broken-commit  
$ git bisect bad  
$ git checkout working-commit  
$ git bisect good
```

Git will checkout the commit in between the two you've provided, and ask you to test it and determine if its working or broken



---

# Ch 7: bisect

---

If it's working, run

```
$ git bisect good
```

If it's broken, run

```
$ git bisect bad
```

Either way, Git will use that information to determine the best commit to test next

---

# Ch 7: bisection

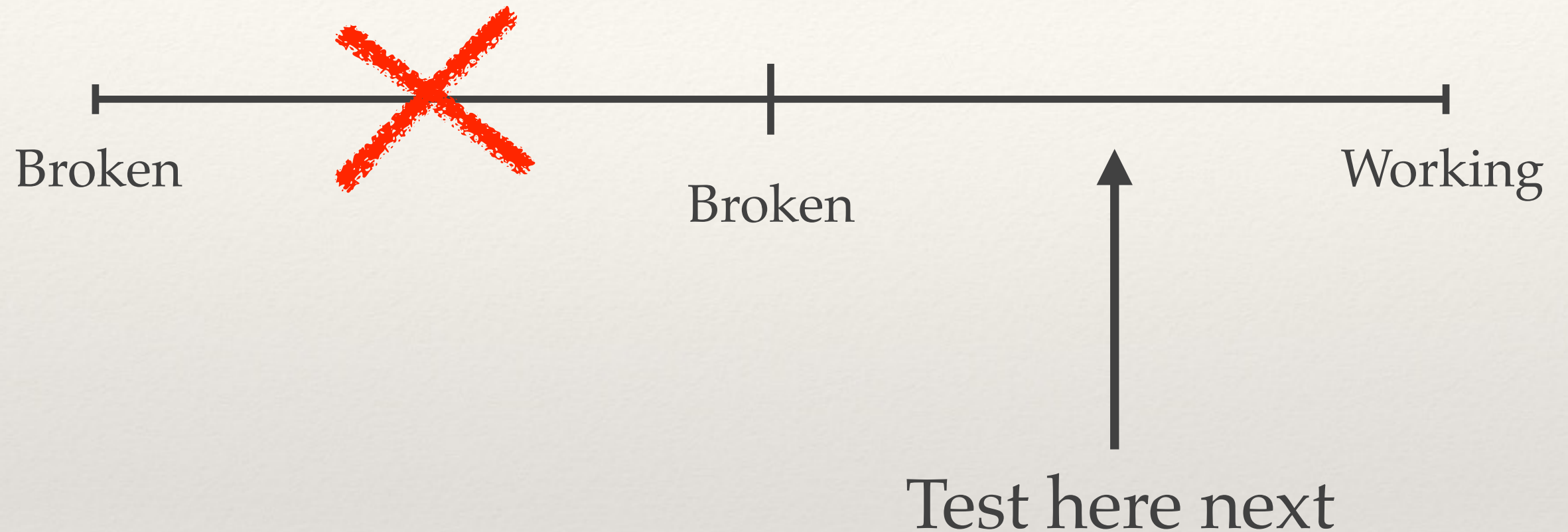
---



---

# Ch 7: bisection

---

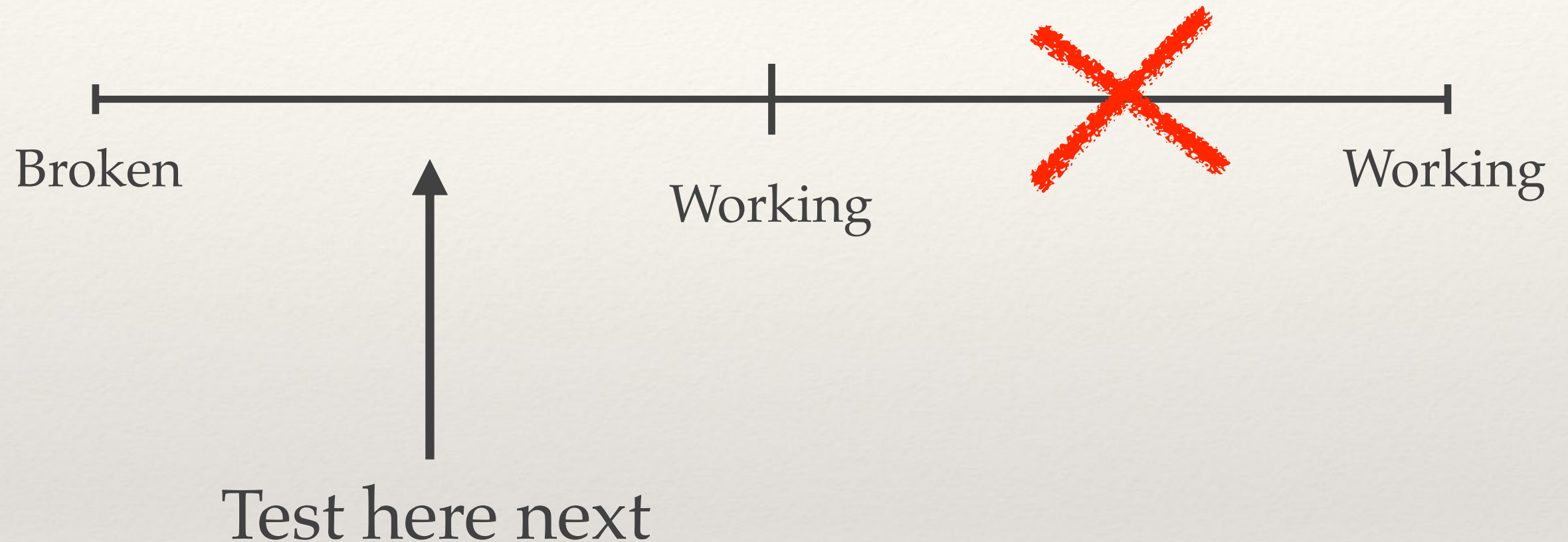




---

# Ch 7: bisection

---



And keep going recursively....

---

# Ch 7: bisect

---

If you have an automated test, it's even faster!

```
$ git bisect run my_test.sh
```

With that, Git can  
test, checkout, test, checkout, test  
until it finds the commit  
that caused the failure

---

# Table of Contents

---

Preface

status, show

Ch 1

blame

Ch 2

cherry-pick

Ch 3

et

Ch 4

rebase

Ch 5

reflog

Ch 6

squashing & splitting

Ch 7

bisect

**DONE!**



---

# There's so much more...

---

```
$ git help rebase
```

<http://git-scm.com/doc>

<http://help.github.com>

# Head First Git

A Brain-Friendly Guide



Editing has never  
been so enlightening



Compatible  
with all kinds  
of text, not  
just software



Track down issues  
at their source



A learner's guide to Git

Discover the  
freedom in  
branching,  
forking,  
& merging



Learn how  
to travel  
through  
time



Change history without  
tying yourself in knots

David Baumgold

---

# Any questions?

---

David Baumgold  
@singingwolfboy

*As a reminder, we covered*

blame cherry-pick reset rebase  
reflog squash split bisect

Book: [davidbaumgold.com/book](http://davidbaumgold.com/book)

Slides: [bit.ly/git-pydx-2016](http://bit.ly/git-pydx-2016)